Assignment 2

Student Number: 19200098

Importing Libraries

Importing necessary libraries from the packages - Sklearn, BeautifulSoup, NLTK, Requests, Regular Expression (RE)

In [1]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt


from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.naive_bayes import MultinomialNB

import warnings
warnings.filterwarnings("ignore")
```

Initializing the URL to extract the content from

Inside the URL we have 7 different categories from where we need to extract the data. Setting of 3 categories of interest is done which are **Fashion, Gym and Hair Salons**.

In [2]:
```
#setting URL
URL = "http://mlg.ucd.ie/modules/yalp/"

#storing the content of the page
page = requests.get(URL)

#Fetching all category names
soup = BeautifulSoup(page.content, 'html.parser')
results = soup.find(id='all')
```

In [3]:
```
categories = results.find_all('h4')

#Fetching category names - Fashion, Gym and Hair Salons and printing their URL
s
for category in categories:
    if category.find('a')['href'] == "fashion_list.html":
        URL_fashion = URL + "fashion_list.html"
    elif category.find('a')['href'] == "gym_list.html":
        URL_gym = URL + "gym_list.html"
    elif category.find('a')['href'] == "hair_salons_list.html":
        URL_hair_salons = URL + "hair_salons_list.html"

print(URL_fashion)
print(URL_gym)
print(URL_hair_salons)
```

```
http://mlg.ucd.ie/modules/yalp/fashion_list.html
http://mlg.ucd.ie/modules/yalp/gym_list.html
http://mlg.ucd.ie/modules/yalp/hair_salons_list.html
```

Part 1: Extraction of 3 categories

When the data is extracted the number of stars given by the review is stored. Now, as we need to classify each review in positive and negative to make it easy to understand what the review indicates, we would convert all the ratings greater than or equal to 3 as positive or 1, and all ratings less than 3 as negative or 0.

Data Extraction for the categories - Fashion, Gym and Hair Salons

In [4]:
```
#Scrapping the web page
def category_scrapping(category_URL):
    list_rating = []
    list_review = []
    page_category = requests.get(category_URL)
    soup_category = BeautifulSoup(page_category.content, 'html.parser')

    category_list = soup_category.find_all('h5')

    for reviews in category_list:
        URL_reviews = reviews.find('a')['href']
        URL_reviews = URL + URL_reviews
        page_reviews = requests.get(URL_reviews)
        soup_reviews = BeautifulSoup(page_reviews.content, 'html.parser')
        reviews_list = soup_reviews.find_all('div', class_='review')
        for review in reviews_list:
            rating = review.find('img', alt=True)
            review_text = review.find('p', class_='review-text')
            list_rating.append(int(rating['alt'].split('-')[0]))
            list_review.append(review_text.text)

    category_data = pd.DataFrame({'rating':list_rating, 'review':list_review})

    category_data.loc[category_data['rating'] <3, 'rating'] = 0
    category_data.loc[category_data['rating'] >=3, 'rating'] = 1

    return category_data
```

Extracting data from Fashion Page

In [5]:
```
fashion_data = category_scrapping(URL_fashion)
fashion_data.head()
```

Out[5]:

	rating	review
0	1	Looking for the best tactical supplies? Look n...
1	0	Stood in line like an idiot for 5 minutes to p...
2	1	Another great store with quality Equipment. Th...
3	1	The Problem with this store is not that they h...
4	1	Great place! We went in at almost closing time...

Extracting data from Gym Page

In [6]:
```
gym_data = category_scrapping(URL_gym)
gym_data.head()
```

Out[6]:

	rating	review
0	1	If you're looking for boxing in the East Valle...
1	0	I was really excited to try a fun workout rout...
2	0	I was interested in taking a boxing bootcamp c...
3	1	I worked out at 1 on 1 boxing for a bout 6 mon...
4	1	This place literally KICKED my butt every. sin...

Extracting data from Hair Salons Page

In [7]:
```
hair_salons_data = category_scrapping(URL_hair_salons)
hair_salons_data.head()
```

Out[7]:

	rating	review
0	1	One of the best barbershops I've been to, with...
1	1	Took my son in for a haircut. Barber was great...
2	0	Walked in, said hi. The only barber in there d...
3	0	I came here 10 minutes before 9am to get a hai...
4	1	Great haircut. No fuss no muss, I asked for la...

Part 2: Processing upon each dataset

a. Pre-processing steps to create numeric representation

The reviews in each category is in text format. To normalize text data we perform the following pre-processing steps as in text analytics:

- Tokenizing
- Lowering case
- Stopwords removal
- Stemming
- Lemmatization

In [8]:
```
#Normalization of Data
def data_normalization(data):

    #splitting into tokens using regex
    tokens=[]
    tokenizer = RegexpTokenizer(r'\w+')
    for v in data['review']:
        v = re.sub(r"[0-9]+","",v)
        tokens.append(tokenizer.tokenize(v))

    #convert tokens into lower case
    tokens_lowercase=[]
    for t in tokens:
        tokens_lowercase.append(list(w.lower() for w in t))

    #remove stop words
    stop_words = list(stopwords.words('english'))
    normalized_tokens=[]
    for t in tokens_lowercase:
        temp = set(t) - set(stop_words)
        normalized_tokens.append(list(temp))

    #stemming words
    stemmer = PorterStemmer()
    for tokens in normalized_tokens:
        for w in tokens:
            w = stemmer.stem(w)

    #lemmatizing words
    lemmatizer = WordNetLemmatizer()
    for tokens in normalized_tokens:
        for w in tokens:
            w = lemmatizer.lemmatize(w)

    #joining words to make string
    normalized_words = []
    for words in normalized_tokens:
        normalized_words.append(' '.join(words))

    data['review'] = normalized_words
```

Normalizing data from Fashion Category

In [9]:
```
data_normalization(fashion_data)
fashion_data.head()
```

Out[9]:

	rating	review
0	1	supplies come best civilian looking emt gear p...
1	0	counter minutes comfortable inventory probably...
2	1	super help helpful staff quality looking find ...
3	1	company guy confident come always welcomed fan...
4	1	purchase went employees time closing rushed kn...

Normalizing data from Gym Category

In [10]:
```
data_normalization(gym_data)
gym_data.head()
```

Out[10]:

	rating	review
0	1	recommend hard limits learned nothing gyms got...
1	0	honor anther trained services oh even group dr...
2	0	listen x monthly return preoccupied mentally d...
3	1	significant recommend always though return pun...
4	1	gloves dreaded mile aside literally buying bes...

Normalizing data from Hair Salons Category

In [11]:
```
data_normalization(hair_salons_data)
hair_salons_data.head()
```

Out[11]:

	rating	review
0	1	comfortable found honestly remember quite best...
1	1	barber exactly often took definitely wanted cu...
2	0	soccer minutes nothing watching front game wal...
3	0	showed minutes new haircut saturday hours eith...
4	1	exactly layers fuss ordered returning muss bea...

Combining data from all categories together

In [12]:
```
all_categories = pd.DataFrame()

print('Size of individual categories')
print(fashion_data.shape)
print(gym_data.shape)
print(hair_salons_data.shape)

all_categories = all_categories.append(fashion_data, ignore_index=True)
all_categories = all_categories.append(gym_data, ignore_index=True)
all_categories = all_categories.append(hair_salons_data, ignore_index=True)

print('Size of all categories when combined into single dataset')
print(all_categories.shape)
```

```
Size of individual categories
(2000, 2)
(2000, 2)
(2000, 2)
Size of all categories when combined into single dataset
(6000, 2)
```

b. Classification model to distinguish between "positive" and "negative" reviews

I have chosen **Naive Bayes** classifier to distinguish between the rating labels of the dataset. For this purpose, the dataset has to be divided into training and test data. The model would be trained on the basis of training data and will be tested on the test data.

Split data into training and test format

In [13]:
```
# individual category data divided into training and test set
fashion_data_x_train, fashion_data_x_test, fashion_data_y_train, fashion_data_
y_test = train_test_split(fashion_data['review'], fashion_data['rating'], test
_size=0.33, random_state=42)
gym_data_x_train, gym_data_x_test, gym_data_y_train, gym_data_y_test = train_t
est_split(gym_data['review'], gym_data['rating'], test_size=0.33, random_state
=42)
hair_salons_data_x_train, hair_salons_data_x_test, hair_salons_data_y_train, h
air_salons_data_y_test = train_test_split(hair_salons_data['review'], hair_sal
ons_data['rating'], test_size=0.33, random_state=42)
```

Vectorizing the x-train and x-test data

The tfidf verctorizer converts the raw documents into Tf-Idf features, which would further help us predict and evaluate the model based on the class label of the dataset.

In [14]:
```
#creating object of tfidf vectorizer
vectorizer = TfidfVectorizer()

#fitting and transforming all category data individually

fashion_data_x_train_vectorized = vectorizer.fit_transform(fashion_data_x_trai
n)
fashion_data_x_test_vectorized = vectorizer.transform(fashion_data_x_test)

gym_data_x_train_vectorized = vectorizer.fit_transform(gym_data_x_train)
gym_data_x_test_vectorized = vectorizer.transform(gym_data_x_test)

hair_salons_data_x_train_vectorized = vectorizer.fit_transform(hair_salons_dat
a_x_train)
hair_salons_data_x_test_vectorized = vectorizer.transform(hair_salons_data_x_t
est)
```

Build the model
-The **Naive Bayes** model is built on the basis of provided values of training data

In [15]:
```
def build_naive_bayes(category_data_x_train, category_data_y_train):
    model = MultinomialNB()
    model.fit(category_data_x_train, category_data_y_train)
    return model
```

In [16]:
```
#Each category training data is passed to built Naive Bayes model and predicti
ons are made

fashion_data_model = build_naive_bayes(fashion_data_x_train_vectorized, fashio
n_data_y_train)
fashion_data_predicted_value = fashion_data_model.predict(fashion_data_x_test_
vectorized)

gym_data_model = build_naive_bayes(gym_data_x_train_vectorized, gym_data_y_tra
in)
gym_data_predicted_value = gym_data_model.predict(gym_data_x_test_vectorized)

hair_salons_data_model = build_naive_bayes(hair_salons_data_x_train_vectorized
, hair_salons_data_y_train)
hair_salons_data_predicted_value = hair_salons_data_model.predict(hair_salons_
data_x_test_vectorized)
```

c. Predictions of the classification model using an appropriate evaluation strategy

The evaluation is performed for each category of data individually based on **confusion matrix and classification report**, which gives measures for each as follows:

In [17]:
```
accuracy_values = pd.DataFrame(index=['Fashion', 'Gym', 'Hair Salon'], columns
=['Fashion', 'Gym', 'Hair Salon'])
```

In [18]:
```
#evaluation of each category data based on predicted value and test data
def model_evaluation(predicted_value, true_value, train_category, test_categor
y):
    print(confusion_matrix(predicted_value, true_value))
    print(classification_report(predicted_value, true_value))
    accuracy_values[train_category][test_category] = accuracy_score(predicted_
value, true_value)
```

In [19]:
```
print("-------------------Evaluation of Categories-------------------")
print()
print("_________________Fashion Data_________________")
model_evaluation(fashion_data_predicted_value, fashion_data_y_test, 'Fashion',
'Fashion')
print("_________________Gym Data_________________")
model_evaluation(gym_data_predicted_value, gym_data_y_test, 'Gym', 'Gym')
print("_________________Hair Salons Data_________________")
model_evaluation(hair_salons_data_predicted_value, hair_salons_data_y_test, 'H
air Salon', 'Hair Salon')
```

```
-------------------Evaluation of Categories-------------------

_________________Fashion Data_________________
[[ 58   0]
 [187 415]]
              precision    recall  f1-score   support

           0       0.24      1.00      0.38        58
           1       1.00      0.69      0.82       602

    accuracy                           0.72       660
   macro avg       0.62      0.84      0.60       660
weighted avg       0.93      0.72      0.78       660

_________________Gym Data_________________
[[ 83   1]
 [106 470]]
              precision    recall  f1-score   support

           0       0.44      0.99      0.61        84
           1       1.00      0.82      0.90       576

    accuracy                           0.84       660
   macro avg       0.72      0.90      0.75       660
weighted avg       0.93      0.84      0.86       660

_________________Hair Salons Data_________________
[[  0   0]
 [142 518]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.78      0.88       660

    accuracy                           0.78       660
   macro avg       0.50      0.39      0.44       660
weighted avg       1.00      0.78      0.88       660
```

Part 3: Performance of each of your three classification models when applied to data from the other two selected categories

From the combined dataset, respective data are picked to assign as training and test data for that category.

In [20]:
```
# all category data
fashion_data_x_train, fashion_data_x_test, fashion_data_y_train, fashion_data_
y_test = train_test_split(all_categories['review'][:2000], all_categories['rat
ing'][:2000], test_size=0.33, random_state=42)
gym_data_x_train, gym_data_x_test, gym_data_y_train, gym_data_y_test = train_t
est_split(all_categories['review'][2001:4000], all_categories['rating'][2001:4
000], test_size=0.33, random_state=42)
hair_salons_data_x_train, hair_salons_data_x_test, hair_salons_data_y_train, h
air_salons_data_y_test = train_test_split(all_categories['review'][4001:6000],
all_categories['rating'][4001:6000], test_size=0.33, random_state=42)
```

Vectorizing the training data

In [21]:
```
#initializing tfidf vectorizer object
vectorizer = TfidfVectorizer()

#Vectorizing the train data for each category

fashion_data_x_train_vectorized = vectorizer.fit_transform(fashion_data_x_trai
n)
fashion_data_x_test_vectorized = vectorizer.transform(fashion_data_x_test)

gym_data_x_train_vectorized = vectorizer.fit_transform(gym_data_x_train)
gym_data_x_test_vectorized = vectorizer.transform(gym_data_x_test)

hair_salons_data_x_train_vectorized = vectorizer.fit_transform(hair_salons_dat
a_x_train)
hair_salons_data_x_test_vectorized = vectorizer.transform(hair_salons_data_x_t
est)
```

Building Naive Bayes model for each category training data and predicting using test data

In [22]:
```
#building the model and predicting using test data

fashion_data_model = build_naive_bayes(fashion_data_x_train_vectorized, fashio
n_data_y_train)
fashion_data_predicted_value = fashion_data_model.predict(fashion_data_x_test_
vectorized)

gym_data_model = build_naive_bayes(gym_data_x_train_vectorized, gym_data_y_tra
in)
gym_data_predicted_value = gym_data_model.predict(gym_data_x_test_vectorized)

hair_salons_data_model = build_naive_bayes(hair_salons_data_x_train_vectorized
, hair_salons_data_y_train)
hair_salons_data_predicted_value = hair_salons_data_model.predict(hair_salons_
data_x_test_vectorized)
```

a. Train a classification model on the data from “Category A”. Evaluate its performance on data from “Category B” and data from “Category C”

Category A - Fashion
Category B - Gym
Category C - Hair Salon

In [23]:
```
print("-------------------Evaluation of Categories-------------------")
print()
print("_________________Fashion Data against Gym Data_________________")
model_evaluation(fashion_data_predicted_value, gym_data_y_test, 'Fashion', 'Gym')
print("_________________Fashion Data against Hair Salons Data_________________")
model_evaluation(fashion_data_predicted_value, hair_salons_data_y_test, 'Fashion', 'Hair Salon')
```

```
-------------------Evaluation of Categories-------------------

_________________Fashion Data against Gym Data_________________
[[ 21  37]
 [161 441]]
              precision    recall  f1-score   support

           0       0.12      0.36      0.18        58
           1       0.92      0.73      0.82       602

    accuracy                           0.70       660
   macro avg       0.52      0.55      0.50       660
weighted avg       0.85      0.70      0.76       660

_________________Fashion Data against Hair Salons Data_________________
[[  9  49]
 [124 478]]
              precision    recall  f1-score   support

           0       0.07      0.16      0.09        58
           1       0.91      0.79      0.85       602

    accuracy                           0.74       660
   macro avg       0.49      0.47      0.47       660
weighted avg       0.83      0.74      0.78       660
```

b. Train a classification model on the data from “Category B”. Evaluate its performance on data from “Category A” and data from “Category C”.

Category A - Fashion
Category B - Gym
Category C - Hair Salon

In [24]:
```
print("-------------------Evaluation of Categories-------------------")
print()
print("_________________Gym Data against Fashion Data_________________")
model_evaluation(gym_data_predicted_value, fashion_data_y_test, 'Gym', 'Fashio
n')
print("_________________Gym Data against Hair Salons Data_________________")
model_evaluation(gym_data_predicted_value, hair_salons_data_y_test, 'Gym', 'Ha
ir Salon')
```

```
-------------------Evaluation of Categories-------------------

_________________Gym Data against Fashion Data_________________
[[ 27  47]
 [218 368]]
              precision    recall  f1-score   support

           0       0.11      0.36      0.17        74
           1       0.89      0.63      0.74       586

    accuracy                           0.60       660
   macro avg       0.50      0.50      0.45       660
weighted avg       0.80      0.60      0.67       660

_________________Gym Data against Hair Salons Data_________________
[[ 18  56]
 [115 471]]
              precision    recall  f1-score   support

           0       0.14      0.24      0.17        74
           1       0.89      0.80      0.85       586

    accuracy                           0.74       660
   macro avg       0.51      0.52      0.51       660
weighted avg       0.81      0.74      0.77       660
```

c. Train a classification model on the data from “Category C”. Evaluate its performance on data from “Category A” and data from “Category B”.

Category A - Fashion
Category B - Gym
Category C - Hair Salon

In [25]:
```
print("-------------------Evaluation of Categories-------------------")
print()
print("_________________Hair Salons Data against Fashion Data_________________")
model_evaluation(hair_salons_data_predicted_value, fashion_data_y_test, 'Hair
 Salon', 'Fashion')
print("_________________Hair Salons Data against Gym Data_________________")
model_evaluation(hair_salons_data_predicted_value, gym_data_y_test, 'Hair Salo
n', 'Gym')
```

```
-------------------Evaluation of Categories-------------------

_________________Hair Salons Data against Fashion Data_________________
[[  2   1]
 [243 414]]
              precision    recall  f1-score   support

           0       0.01      0.67      0.02         3
           1       1.00      0.63      0.77       657

    accuracy                           0.63       660
   macro avg       0.50      0.65      0.39       660
weighted avg       0.99      0.63      0.77       660

_________________Hair Salons Data against Gym Data_________________
[[  2   1]
 [180 477]]
              precision    recall  f1-score   support

           0       0.01      0.67      0.02         3
           1       1.00      0.73      0.84       657

    accuracy                           0.73       660
   macro avg       0.50      0.70      0.43       660
weighted avg       0.99      0.73      0.84       660
```

Accuracy Values of each category against other categories

In [26]:
```
accuracy_values
```

Out[26]:

	Fashion	Gym	Hair Salon
Fashion	0.716667	0.598485	0.630303
Gym	0.7	0.837879	0.725758
Hair Salon	0.737879	0.740909	0.784848

Observations:

Evaluations

- All categories when tested against its own data gives highest accuracy.
- Fashion data gives less accuracy when tested against Gym data, but vice versa is not true.
- Gym data and Hair Salon data gives equivalent accuracy scores when tested against their respective opposite categories.

Confusion Matrix and Classification Report

- In confusion matrix, True Positives and True negatives (bottom row of the matrix) are higher as we are getting higher accuracy score values.
- In most cases, the precision and recall scores are good in terms of 'positive' reviews, i.e class 1; which indicates that there are more positive reviews than negative. Surprisingly, these are maintained even when we test the model with a different category data.

References:

1. https://realpython.com/beautiful-soup-web-scraper-python/#decipher-the-information-in-urls (https://realpython.com/beautiful-soup-web-scraper-python/#decipher-the-information-in-urls)

In [ ]: